МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные системы управления базами данных»

Тема: Поиск подозрительных комментаторов в пабликах ВК

Студент гр. 8303 _____ Журбин К.А.

Студент гр. 8303 _____ Курлин Н.Н.

Студент гр. 8303 _____ Удод М.Н.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2021

ЗАДАНИЕ

НА ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

Студенты

Журбин К.А.

Журбин Н.Н.

Удод М.Н.

Группа 8303

Тема работы: Поиск подозрительных комментаторов в пабликах ВК

Исходные данные:

Веб-инструмент для выгрузки данных о комментариях в сообществах, данных о комментаторах, сборка социальных графов для визуализации, аггрегации и определения подозрительных персонажей (спамеры, боты, троли).

Содержание пояснительной записки:

«Содержание»

«Введение»

«Качественные требования к решению»

«Сценарии использования»

«Модель данных»

«Разработанное приложение»

«Выводы»

«Приложения»

«Использованные материалы»

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студент гр. 8303	_____	Журбин К.А.
Студент гр. 8303	_____	Курлин Н.Н.
Студент гр. 8303	_____	Удод М.Н.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В данной работе реализуется приложение для обнаружения нежелательных комментариев в группе сети Вконтакте. Для этого в приложении доступны для просмотра комментарии пользователя, их число, отношения числа комментариев к числу лайков и другая информация, которая может помочь владельцу группы определить нежелательные комментарии.

Приложение имеет Web-интерфейс с использованием библиотеки React, серверная часть реализована на языке JavaScript с использованием библиотеки express. В качестве СУБД используется mongodb.

SUMMARY

In this work, an application is implemented for detecting unwanted comments in a Vkontakte network group. To do this, user comments, their number, the ratio of the number of comments to the number of likes, and other information that can help the group owner identify unwanted comments are available for viewing in the application.

The application has a Web interface using the React library, the server side is implemented in JavaScript using the express library. Mongodb is used as a DBMS.

Оглавление

ВВЕДЕНИЕ

Цель данной работы — реализация приложения, которое поможет владельцу группы находить нежелательные комментарии. Для этого было создано web-приложение, позволяющее просматривать список комментариев, статистику по группе и информацию о пользователях.

1. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ

Требуется разработать приложение с использованием MongoDB, дающее доступ к информации о комментариях и комментаторах, социальным графам и определения подозрительных персонажей.

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет UI

1. Макет приложения

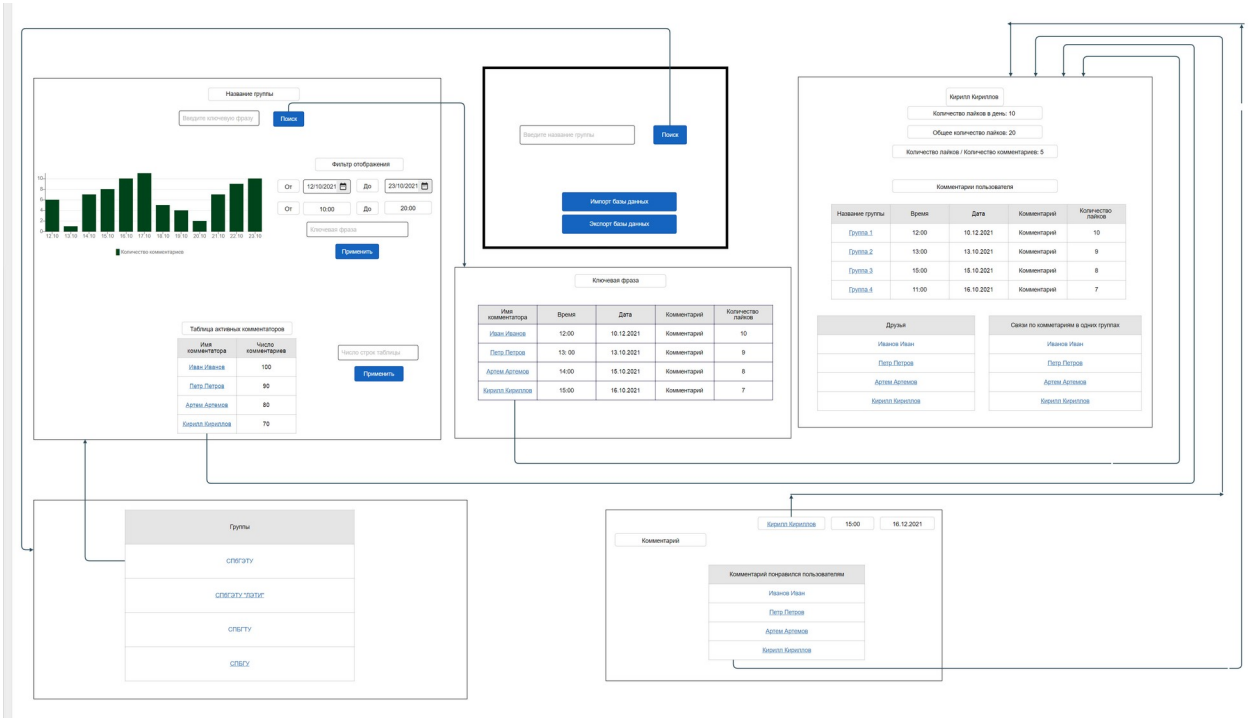

Рисунок 1. Макет основного приложения

2. Макет отладочного приложения

Рисунок 2. Макет отладочного приложения

2.2. Сценарии использования

Единственная роль в системе — владелец группы.
1. Просмотр самых активных комментаторов
Основной сценарий:

1. Пользователь указывает название группы
2. Из списка предложенных групп пользователь выбирает необходимую
3. Пользователь попадает на страницу группы, в которой отображается список активных комментаторов
4. Пользователь указывает максимальное число активных комментаторов
5. Пользователь видит активных комментаторов
6. Пользователь переходит на страницу интересующего комментатора (переход к сценарию использования "Поиск комментариев по ключевым словам")

Альтернативный сценарий:

- Группа не найдена
- Комментариев в группе нет

2. Поиск комментариев по ключевым словам
Основной сценарий:

1. Пользователь указывает название группы
2. Из списка предложенных групп пользователь выбирает необходимую
3. Пользователь попадает на страницу группы, на которой видит поисковую строку
4. Пользователь вводит ключевую фразу в поисковую строку
5. Пользователь видит список комментариев с указанной фразой

6. Пользователь переходит на страницу интересующего комментатора (переход к сценарию использования "Поиск комментариев по ключевым словам")

Альтернативный сценарий:

- Группа не найдена
- Комментариев в группе нет

3. Просмотр страницы комментатора

Основной сценарий:

1. Пользователь переходит на страницу комментаторам
2. Пользователь видит число комментариев в единицу времени
3. Пользователь видит общее число лайков, поставленных на комментарии пользователя(комментатора) и число (кол-во всех лайков)/(кол-во всех комментариев)
4. Пользователь видит список комментариев в этой и других группах
5. Пользователь видит социальный граф (таблицу) с другими комментаторами (комментаторы связываются, если они прокомментировали в одном паблике)
6. Пользователь видит социальный граф (таблицу) с другими комментаторами (комментаторы связываются, если они находятся в друзьях друг у друга)

Альтернативный сценарий:

- Профиль комментатора закрыт, друзей просмотреть невозможно -> социальный граф по друзьям построить невозможно
- Комментатор не имеет друзей -> социальный граф по друзьям построить невозможно

4. Просмотр статистики группы

Основной сценарий:

1. Пользователь указывает название группы
2. Из списка предложенных групп пользователь выбирает необходимую
3. Пользователь попадает на страницу группы
4. Пользователь видит диаграмму числа комментариев по временной шкале
5. Пользователь видит диаграмму числа лайков по временной шкале

Альтернативный сценарий:

- Группа не найдена
- Комментариев в группе нет
- Лайков в группе нет

5. Просмотр лайков на комментарий

Основной сценарий:

1. На странице с ключевой фразой в комментарии пользователь нажимает на число, отображающее количество лайков
2. Пользователь попадает на страницу с информацией о комментарии
3. Пользователь видит список людей, лайкнувших комментарий, автора комментария, время и дату публикации комментария

6. Импорт

Основной сценарий:

1. Пользователь переходит на главную страницу приложения
2. Пользователь нажимает на «Импорт»
3. Пользователь выбирает файл, содержащий информацию, необходимую для импорта

7. Экспорт

Основной сценарий:

1. Пользователь переходит на главную страницу приложения
2. Пользователь нажимает на «Экспорт»

2.3. Преобладание чтения или записи

Из всех перечисленных сценариев только «Экспорт» может записывать информацию в базу данных, а значит для данного решения чтение преобладает над записью.

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных

3.1.1. Графическое представление

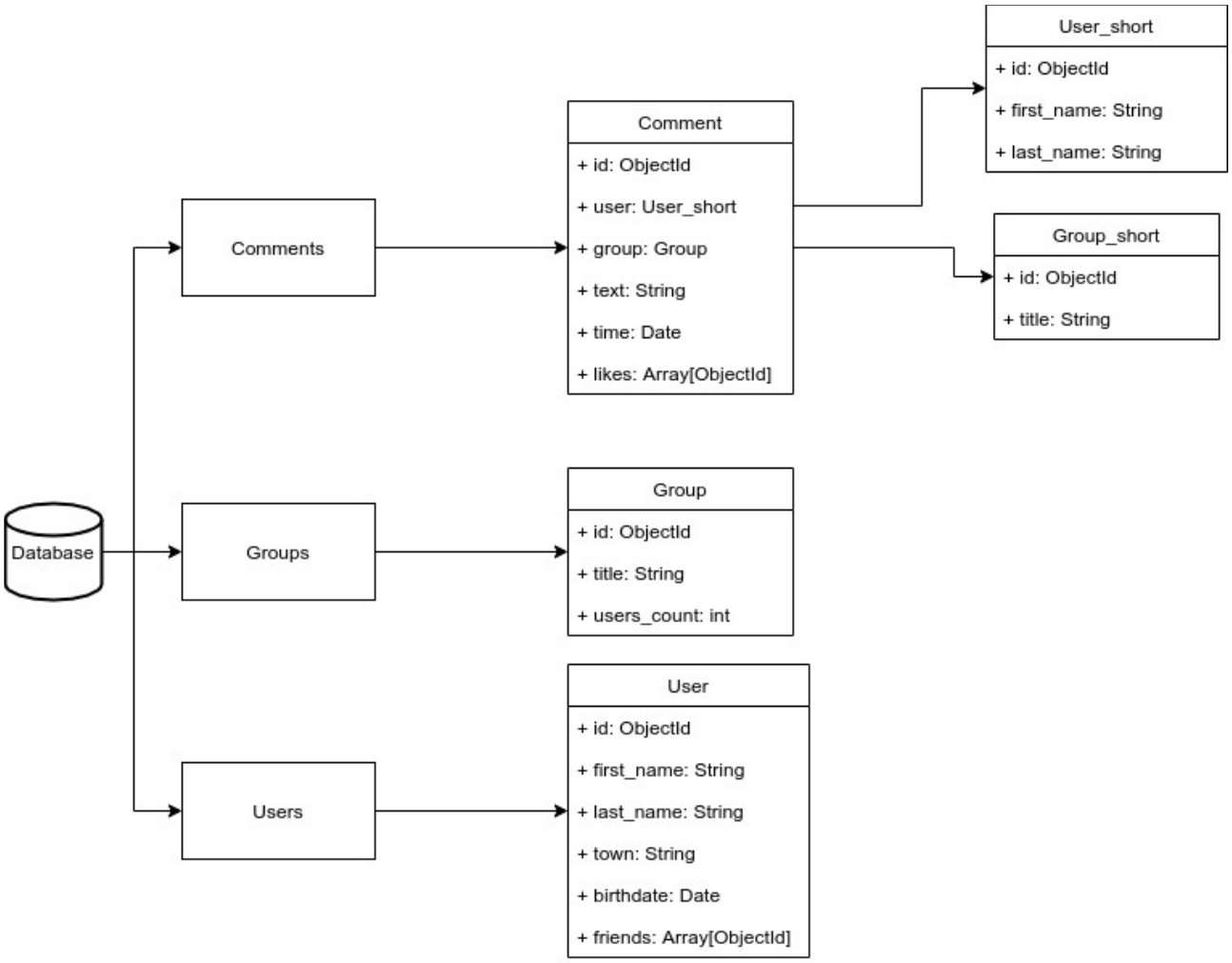

Рисунок 3. Нереляционная модель данных

3.1.2. Описание назначений коллекций, типов данных и сущностей

БД содержит 3 коллекции "Comments", "Groups" и "Users"
Users содержит массив объектов типа **User**

- _id - уникальный идентификатор пользователя
- first_name - имя пользователя
- last_name - фамилия пользователя
- town - город
- birtdate - день рождения
- friend - список id друзей

Groups содержит массив объектов типа **Group**

- _id - уникальный идентификатор группы
- title - название группы
- user_count - число пользователей в группе

Comments содержит массив объектов типа **Comment**

- _id - уникальный идентификатор комментария
- user - информация о пользователе, написавшем комментарий, объект типа **User_short**
- _id - уникальный идентификатор пользователя
- first_name - имя пользователя
- last_name - фамилия пользователя
- group - информация о группе, в которой оставлен комментарий, объект типа **Group_short**
- _id - уникальный идентификатор группы
- title - название группы
- text - текст комментария
- time - время, в которое был создан комментарий
- likes - id пользователей, поставивших лайк

3.1.3. Оценка удельного объема информации, хранимой в модели

Коллекция Users:
Найдем объем для одного пользователя (тип User):

- _id - ObjectId, 8 байт
- first_name - String, пусть максимальная длина имени - 15 символов, 15 байт
- last_name - String, пусть максимальная длина фамилии - 15 символов, 15 байт
- town - String, пусть макисмальная длина города - 25 символов, 25 байт
- birtdate - Date, 8 байт
- friend - Array[ObjectId], пусть среднее число друзей - Nf, 8*Nf

Итого: Vu = 8 + 15 + 15 + 25 + 8 + 8Nf = $71+8$Nf байт

Коллекция **Groups**: Найдем объем для одной группы (тип **Group**):

- _id - ObjectId, 8 байт
- title - String, пусть максимальная длина названия - 40 символов, 40 байт
- user_count - Int, 8 байт

Итого: Vg = 8 + 40 + 8 = 56 байт

Коллекция **Comments**: Найдем объем для одного комментария (тип **Comments**):

- _id - ObjectId, 8 байт

- user - User_short, 38 байт
- _id - ObjectId, 8 байт
- first_name - String, 15 байт
- last_name - String, 15 байт
- group - Group_short, 48 байт
- _id - ObjectId, 8 байт
- title - String, 40 байт
- text - String, предположим, что средняя длина комментария - 255 символов, 255 байт
- time - Date, 8 байт
- likes - Array[ObjectId], пусть среднее число лайков - Nl, 8*Nl байт

Итого: Vc = 8 + 38 + 48 + 255 + 8 + 8*Nl* = *357* + *8*Nl байт

Объем данных, необходимый для хранения Nu пользователей, Ng групп и Nc комментариев:

V(Nu, Ng, Nc) = Nu * Vu + Ng * Vg + Nc * Vc

V(Nu, Ng, Nc) = Nu * (71 + 8 * Nf) + Ng * 56 + Nc * (357 + 8 * Nl)

Для оценки объема предположим, что только каждый сотый пользователь создает группу (Ng = Nu/100) и каждый пользователь оставил в среднем 100 комментариев (Nc = 100*Nu). Так же предположим, что среднее число лайков для комментария - 5 и среднее число друзей - 20.

Тогда для 1000 пользователей: V = 1000 * (71 + 8 * 20) + 10 * 56 + 100000 * (357 + 8 * 5) = 39931560 байт = 4.76 мбайт

Вычисление чистого объема данных

Коллекции **User** и **Group** остаются без изменений: Vu = 71+8*Nf байт Vg = 56 байт

В коллекции **Comments** не цчитываются поля user и group: Vc = 8 + 255 + 8 + 8*Nl* = *271* + *8*Nl байт Для 1000 пользователей при тех же допущениях получим: V_clean = 1000 * (71 + 8 * 20) + 10 * 56 + 100000 * (271 + 8 * 5) = 31331560 байт = 3.7 мбайт Отношение v_clean/v = 0.784

3.1.4. Запросы к модели

В треугольных скобках указаны параметры запроса `<example>`, n — число пользователей, p — число групп, q — число комментариев, l — число лайков, f — число друзей

- Поиск группы по названию

```
db.groups.find({title: /<title>/})
```

Сложность - O(p)

Используется коллекций — 1

- Поиск комментария по ключевой фразе

```
db.comments.aggregate([
    {$match: {text: /<key>/}},
    {$addFields: { likes_count: { $size: "$likes" } }},
    {$project: {likes: 0}}
])
```

Сложность - O(q)
Используется коллекций — 1

- Получение пользователей, лайкнувших комментарий

```
db.comments.aggregate([
    {$match: {_id: 0}},
    {$project: {likes: 1}},
    {$lookup: {
        from: "users",
        localField: "likes",
        foreignField: "_id",
        as: "likes"
    }}
])
```

Сложность - O(nql)
Используется коллекций — 2

- Получение наиболее активных комментаторов в группе

```
db.comments.aggregate([
    {$match: {"group._id": <group_id>}},
    {$group: {
        _id: "$user._id",
        first_name: { $first: "$user.first_name"},
        last_name: { $first: "$user.last_name"},
        count: { $sum: 1 }
    }},
    {$sort: {count: -1}},
    {$limit: <rows_count>}
])
```

Сложность - O(q)
Используется коллекций — 1

- Получение комментариев пользователя

```
db.comments.aggregate([
    {$match: {"user._id": <user_id>}},
    {$addFields: { likes_count: { $size: "$likes" } }},
    {$project: {likes: 0, user: 0}}
])
```

Сложность - O(q)
Используется коллекций — 1

- Получение друзей пользователя

```
db.users.aggregate([
    {$match: {"_id": <user_id>}},
    {$lookup: {
        from: "users",
        localField: "friends",
        foreignField: "_id",
        as: "friends"
    }},
    {$project: {friends: 1}}
])
```

Сложность - O(nf)
Используется коллекций — 1

- Получение связей по комментариям в одной группе

```
db.comments.distinct("user", {"group._id": {
    $in: db.comments.distinct("group._id", {"user._id": <user_id>})
},
    "user._id": {$ne: <user_id>}
})
```

Сложность - $O(q^2)$
Используется коллекций — 1
Число запросов — 2

- Количество лайков пользователя

```
db.comments.count({
    likes: <user_id>
})
```

Сложность - O(q)
Используется коллекций — 1

- Количество комментариев пользователя

```
db.comments.count({
    "user._id": <user_id>
})
```

Сложность - O(q)
Используется коллекций — 1

- Данные для диаграммы количества комментариев по времени

```
db.comments.aggregate([
    {
        $match: {
            _id: <group_id>
        }
    },
    {
        $lookup: {
```

```
            from: "comments",
            localField: "_id",
            foreignField: "group._id",
            as: "comments"
        }
    },
    {
        $unwind: "$comments"
    },
    {
        $project: {
            date: { $dateToString: { format: "%Y-%m-%d", date:
"$comments.time" } },
            time: { $dateToString: { format: "%H:%M:%S", date:
"$comments.time" } },
            text: "$comments.text"
        }
    },
    {
        $match: {
            text: {
                $regex: text,
                $options: 'i'
            },
            date: {
                $gte: min_date,
                $lte: max_date
            },
            time: {
                $gte: min_time,
                $lte: max_time
            }
        }
    },
    {
        $group: {
            _id: "$date",
            count_comments: {$sum: 1}
        }
    },
    {
        $sort: {
            _id: 1
        }
    }
])
```

Сложность - $O(q^2)$
Используется коллекций — 1

- DEBUG: список пользователей

```
db.users.aggregate([
     {$addFields: {
        age: {$subtract: [{$year: ISODate()}, {$year: "$birthdate"}]}
     }},
     {$match: {
        first_name: /<first_name>/,
        last_name: /<last_name>/,
        town: /<town>/,
```

```
        age: {$gt: <ageFrom>, $lt: <ageTo>}
     }},
     {$project: {
        first_name: 1,
        last_name: 1,
        birthdate: 1,
        town: 1
     }}
])
```

Сложность - O(n)
Используется коллекций — 1

- DEBUG: список групп

```
db.groups.aggregate([
    {$lookup: {
        from: "comments",
        localField: "_id",
        foreignField: "group._id",
        as: "comments"
    }},
    {$addFields: {
        comments_count: {$size: "$comments"}
    }},
    {$match: {
        title: /<title>/,
        users_count: {$gte: <ucFrom>, $lte: <ucTo>},
        comments_count: {$gte: <ccFrom>, $lte: <ccTo>}
    }},
    {$project: {
        comments: 0
    }}
])
```

Сложность - O(pq)
Используется коллекций — 2

- DEBUG: список комментариев

```
db.comments.aggregate([
    {
        $project: {
            first_name: "$user.first_name",
            last_name: "$user.last_name",
            text: 1,
            date: { $dateToString: { format: "%Y-%m-%d", date: "$time" } },
            time: { $dateToString: { format: "%H:%M:%S", date: "$time" } },
            likes_count: { $size: "$likes"}
        }
    },
    {
        $match: {
            first_name: {
                $regex: first_name,
                $options: 'i'
            },
            last_name: {
```

```
                $regex: last_name,
                $options: 'i'
            },
            text: {
                $regex: text,
                $options: 'i'
            },
            date: {
                $gte: min_date,
                $lte: max_date
            },
            time: {
                $gte: min_time,
                $lte: max_time
            },
            likes_count: {
                $gte: min_likes_count,
                $lte: max_likes_count
            },
        }
    }
])
```

Сложность - O(q)
Используется коллекций — 1

3.2. Реляционная модель данных

3.2.1. Графическое представление

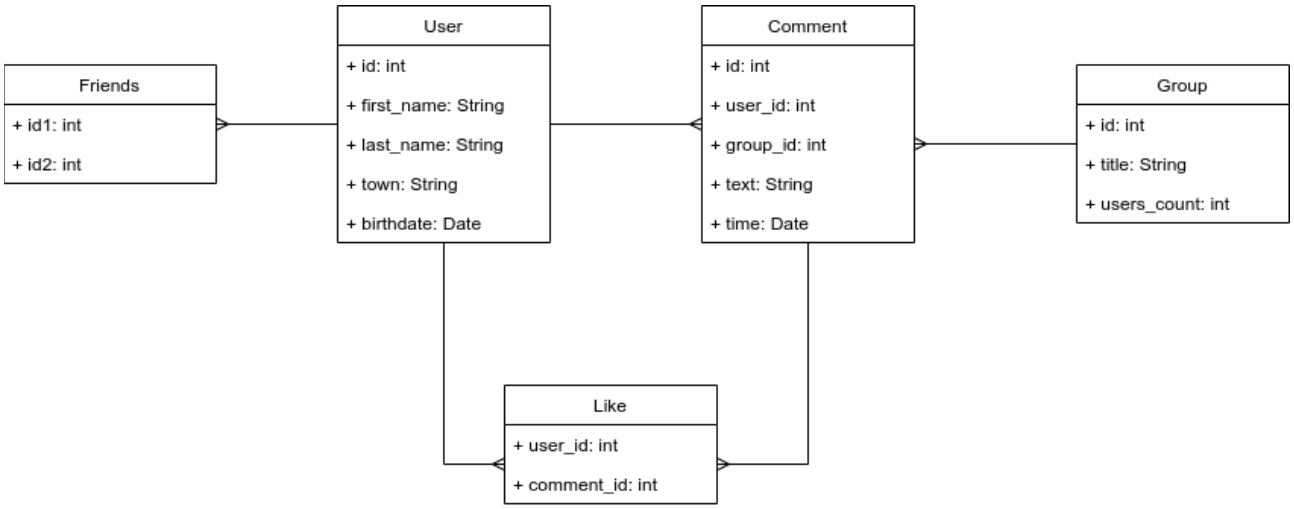

Рисунок 4. Реляционная модель

3.2.2. Описание назначений коллекций, типов данных и сущностей

User - содержит информацию о пользователе

- id - уникальный идентификатор пользователя
- first_name - имя пользователя
- last_name - фамилия пользователя

- town - город
- birtdate - день рождения

Group - содержит информацию о группе
- id - уникальный идентификатор группы
- title - название группы
- user_count - число пользователей в группе

Comment - содержит информацию о комментарии
- _id - уникальный идентификатор комментария
- user_id - id пользователя, оставившего комментарий
- group_id - id группы, в которой был оставлен комментарий
- text - текст комментария
- time - время, в которое был создан комментарий

Like - содержит информацию о лайках
- user_id - id пользователя, поставившего лайк
- comment_id - id комментарий, под которым был поставлен лайк

Friends - содержит информация о друзьях. Пользователь А дружит с пользователем В, если существует строка, содержащая их id.
- id1 - id пользователя, дружащего с пользователем с id == id2
- id2 - id другого пользователя, дружащего с пользователем с id == id1

3.2.3. Оценка удельного объема информации, хранимой в модели

Сущность **User**
- id - Int, 8 байт
- first_name - String, 15 байт
- last_name - String, 15 байт
- town - String, 25 байт
- birtdate - Date, 8 байт

Итого: Vu = 8 + 15 + 15 + 25 + 8 = 71 байт

Сущность **Group**
- id - Int, 8 байт
- title - String, 40 байт
- user_count - Int, 8 байт

Итого: Vg = 8 + 40 + 8 = 56 байт

Сущность **Comment**
- _id - Int, 8 байт
- user_id - Int, 8 байт
- group_id - Int, 8 байт
- text - String, 255 байт
- time - Date, 8 байт

Итого: Vc = 8 + 8 + 8 + 255 + 8 = 287 байт

Сущность **Likes**

- user_id - Int, 8 байт
- comment_id - Int, 8 байт

Итого: Vl = 8 + 8 = 16 байт

Сущность **Friends**
- id1 - Int, 8 байт
- id2 - Int, 8 байт

Итого: Vf = 8 + 8 = 16 байт

Объем данных, необходимый для хранения Nu пользователей, Ng групп и Nc комментариев:
V(Nu, Ng, Nc) = Nu * Vu + Ng * Vg + Nc * Vc + Nu * Nf * Vf + Nc * Nl * Vl
V(Nu, Ng, Nc) = Nu * 71 + Ng * 56 + Nc * 287 + Nu * Nf * 16 + Nc * Nl * 16
Вынесем Nu и Nc за скобки: V(Nu, Ng, Nc) = Nu * (71 + Nf * 16) + Ng * 56 + Nc * (287 + Nl * 16)

Оценим объем для 1000 пользователей, испульзуя те же предположения, что и для нереляционной схемы.
V = 1000 * (71 + 20 * 16) + 10 * 56 + 100000 * (287 + 5 * 16) = 37091560 байт = 4.42 мбайт

Вычисление чистого объема
Для вычисления чистого объема не будем учитывать поля user_id и group_id в сущности **Comment**
Vc = 8 + 255 + 8 = 271 байт

Объем для 1000 польвователей при тех же предположениях: V_clean = 1000 * (71 + 20 * 16) + 10 * 56 + 100000 * (271 + 5 * 16) = 35491560 байт = 4.23 мбайт
Найдем отношение V_clean/V = 35491560/37091560 = 0.95

3.2.4. Запросы к модели

- Поиск группы по названию

```
SELECT * FROM "Groups" WHERE "title" LIKE '%<title>%'
```

Сложность - O(p)
Используется коллекций — 1

- Поиск комментария по ключевой фразе

```
SELECT C.*, count(L.*) as likes_count  FROM "Comments" C
    INNER JOIN "Likes" L on C.id = L.comment_id
```

```
WHERE C.text LIKE '%<key>%'
GROUP BY  C.id
```

Сложность - O(ql)
Используется коллекций — 2

- Получение пользователей, лайкнувших комментарий

```
SELECT DISTINCT U.* FROM "Users" U
    INNER JOIN "Likes" L on U.id = L.user_id
WHERE L.comment_id = <comment_id>
```

Сложность - O(nl)
Используется коллекций — 2

- Получение наиболее активных комментаторов

```
SELECT U.*, count(L.*) as likes_count FROM "Users" U
    INNER JOIN "Likes" L on U.id = L.user_id
    INNER JOIN "Comments" C on L.comment_id = C.id
WHERE C.group_id = <group_id>
GROUP BY U.id
ORDER BY likes_count
```

Сложность - O(nql)
Используется коллекций — 3

- Получение комментариев пользователя

```
SELECT G.title, C.time, C.text, count(L.*) as likes_count FROM "Comments"
C
    INNER JOIN "Groups" G on G.id = C.group_id
    INNER JOIN "Likes" L on C.id = L.comment_id
WHERE C.user_id = <user_id>
GROUP BY G.id, C.id
```

Сложность - O(pql)
Используется коллекций — 3

- Получение друзей пользователя

```
SELECT U2.first_name, U2.last_name FROM "Friends" F
    INNER JOIN "Users" U1 on F.id1 = U1.id
    INNER JOIN "Users" U2 on F.id2 = U1.id
WHERE F.id1 = <user_id>
```

Сложность - $O(n^2f)$
Используется коллекций — 3

- Получение связей по комментариям в одной группе

```
SELECT U.first_name, U.last_name from "Users" U
    INNER JOIN "Comments" C on U.id = C.user_id
WHERE C.group_id IN(SELECT DISTINCT C2.group_id FROM "Comments" C2 WHERE
C2.user_id = 1) AND U.id <> <user_id>
```

Сложность - $O(nq^2)$
Используется коллекций — 3

- Число лайков пользователя

```
SELECT count(*) from "Likes" L
WHERE L.user_id = <user_id>
```

Сложность - O(l)
Используется коллекций — 1

- Количество комментариев пользователя

```
SELECT count(*) from "Comments" C
WHERE C.user_id = <user_id>
```

Сложность - O(q)
Используется коллекций — 1

- Данные для диаграммы количества комментариев по времени

```
SELECT C.time::date, count(*) FROM "Comments" C
WHERE C.time::date > <dateFrom> AND C.time::date < <dateTo> AND
C.time::time > <timeFrom> AND C.time::time < <timeTo> AND C.group_id =
<group_id>
GROUP BY C.time::date
```

Сложность - O(q)
Используется коллекций — 1

- DEBUG: список пользователей

```
SELECT * FROM "Users" U
WHERE date_part('year', age(U.birthdate)) > <ageFrom> AND
date_part('year', age(U.birthdate)) < <ageTo> AND U.first_name LIKE
'%<first_name>%' AND U.last_name LIKE '%<last_name>%' AND U.town LIKE
'%<town>%'
```

Сложность - O(n)
Используется коллекций — 1

- DEBUG: список групп

```
SELECT G.*, count(C.*) as comments_count FROM "Groups" G
    INNER JOIN "Comments" C on G.id = C.group_id
```

```
WHERE G.title LIKE '%<title>%' AND G.users_count > <ucFrom> AND
G.users_count < <ucTo>
GROUP BY G.id
HAVING count(C.*) > <ccFrom> AND count(C.*) < <ccTo>
```

Сложность - O(pq)
Используется коллекций — 2

- DEBUG: список комментариев

```
SELECT C.*, U.first_name, U.last_name, count(L.*) as likes_count FROM
"Comments" C
    INNER JOIN "Likes" L on C.id = L.comment_id
    INNER JOIN "Users" U on U.id = C.user_id
WHERE U.first_name LIKE '%<first_name>%' AND U.last_name LIKE
'%<last_name>%' AND C.text LIKE '%<text>%'
GROUP BY C.id, U.id
HAVING count(L.*) > <lcFrom> AND count(L.*) < <lcTo>
```

Сложность - O(nql)
Используется коллекций — 3

3.3. Сравнение моделей

Поучили, что объем памяти, используемый нереляционной модели больше чем объем реляционной модели. Так же у нереляционной модели меньше отношение "чистого" и "грязного" объема. При этом, сложность запросов в нереляционной модели ниже. В результате, обе модели имеют свои преимущества. При наличии ограничений по памяти следует выбрать или реляционную модель или уменьшить число дублирования в нереляционной. При высоких требованиях к скорости работы следует выбрать нереляционную модель.

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1. Краткое описание

Разработанное веб-приложение хранит список комментариев, пользователей, их написавших, и группах. Открыв страницу своей группы пользователь может увидеть диаграмму зависимости числа комментариев от времени и таблицу наиболее активных комментаторов. Перейдя на страницу комментатора, пользователь может увидеть такие параметры, как количество лайков, поставленных на комментарии пользователя и отношения числа лайков к числу комментариев. Так же на данной странице отображаются сами комментарии.

Просмотрев данную информацию, пользователь может сделать выводы о том, является ли человек нежелательным лицом. Для продолжения поисков пользователь может перейти на страницы пользователей, связанных с текущим в социальном графе.

4.2. Схема экранов приложения

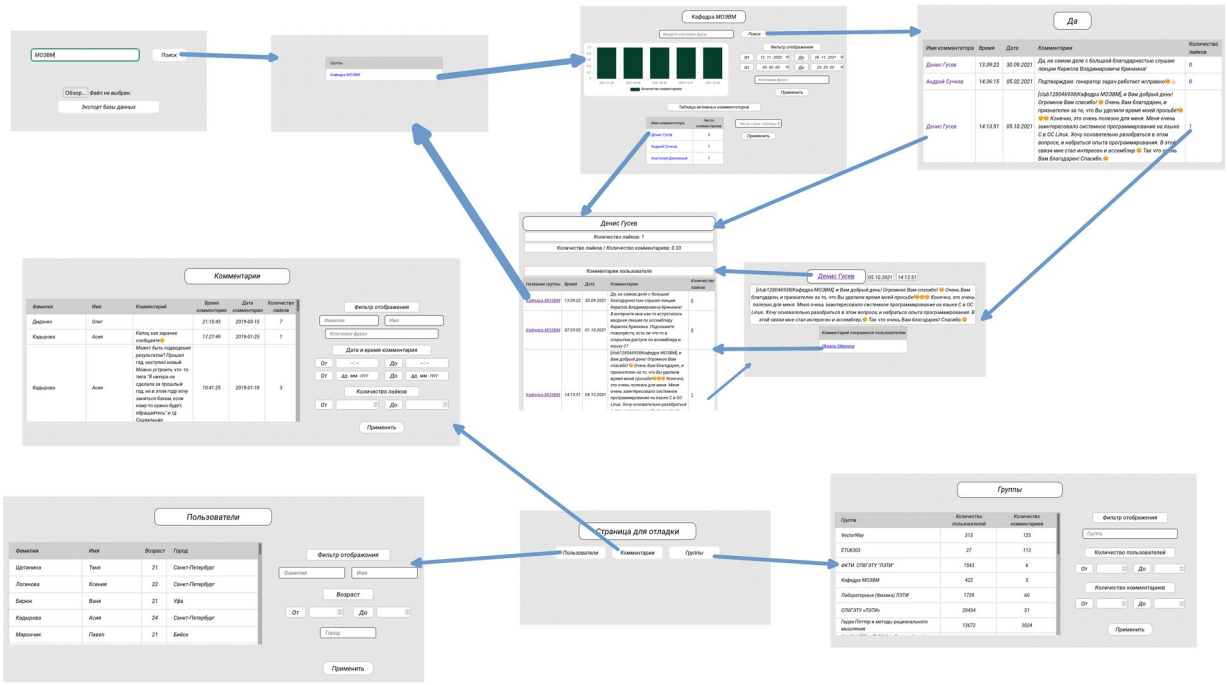

Рисунок 5. Схема приложения

4.3. Использованные технологии

СУБД: MongoDB

Сервер: JavaScript, NodeJS, ExpressJS

Клиент: JavaScript, HTML, CSS, React

4.4. Ссылка на приложение

GitHub: https://github.com/moevm/nosql2h21-vk-comments

5. ВЫВОДЫ

5.1. Достигнутые результаты

В ходе выполнения работы было разработано приложение для поиска подозрительных комментаторов в пабликах ВК. В качестве СУБД используется MongoDB, для данной задачи было проведено сравнение нереляционной и реляционной модели.

5.2. Недостатки и пути для улучшения полученного решения

Для получения данных, используемых в приложении, использовалось VK Api. Из-за его ограничений, была получена информация об относительно небольшом количестве групп, комментариев и пользователей. Для увеличения их числа следует доработать программу их получения таким образом, что бы при достижении ограничении программа останавливалась до того момента, когда ограничения будут сняты.

Еще одним недостатком можно назвать табличное отображение социального графа. Его отображение в виде графа может оказаться более наглядным. Так же при подобной реализации будет возможно просматривать связи на расстоянии большем, чем единица без дополнительных переходов.

5.3. Будущее развитие решения

В дальнейшем может быть реализована версия для мобильных платформ, так как многие владельцы групп могут заниматься администрированием именно с них.

Так же могут быть добавлены функции взаимодействия с комментаторами, такие как запрет комментирования, отправка личного сообщения и другие.

6. ПРИЛОЖЕНИЯ

6.1. Документация по сборке и развертыванию приложения

Для запуска приложения необходимо:

1. Склонировать репозиторий, расположенный по адресу https://github.com/moevm/nosql2h21-vk-comments
2. Собрать приложение командой docker-compose build —no-cache
3. Запустить приложение командой docker-compose up
4. Перейти на адрес http://localhost:3000

6.2. Снимки экрана приложения

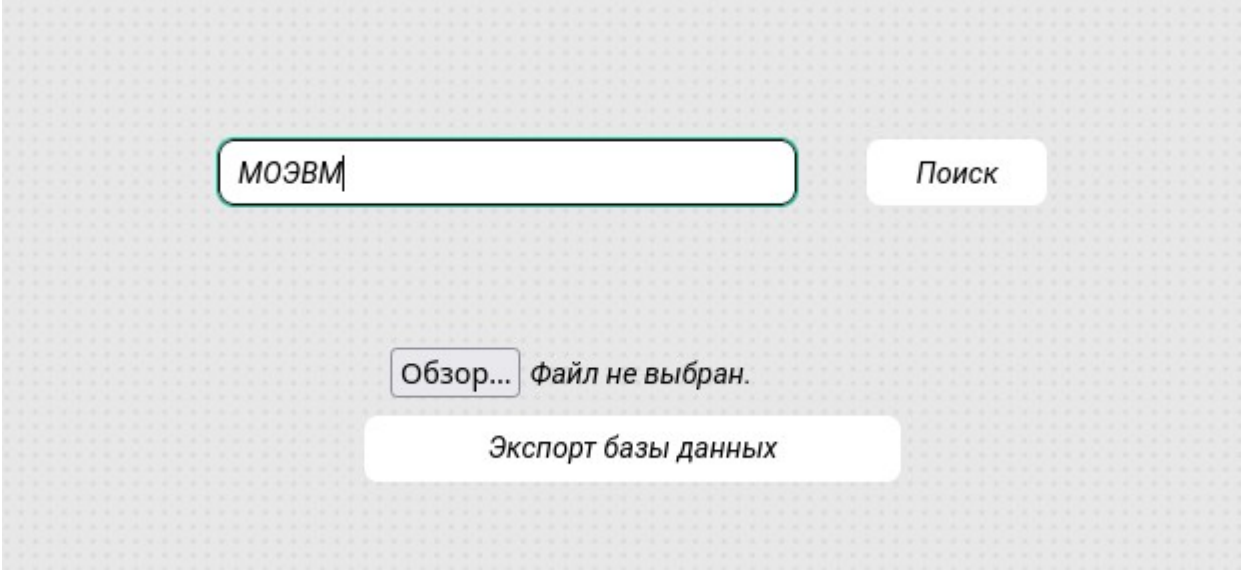

Рисунок 6. Главный экран

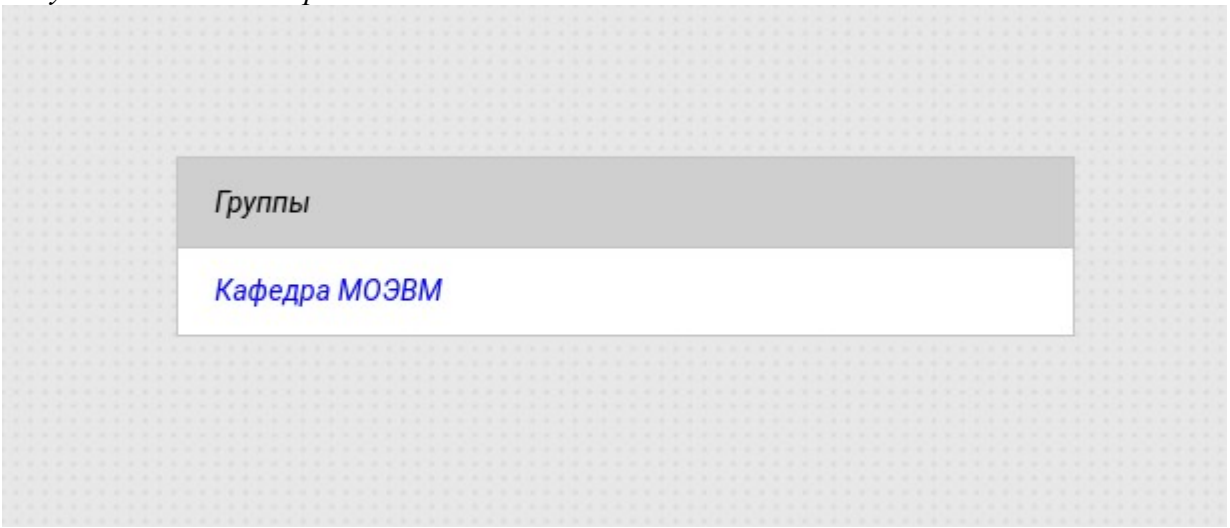

Рисунок 7. Результат поиска группы

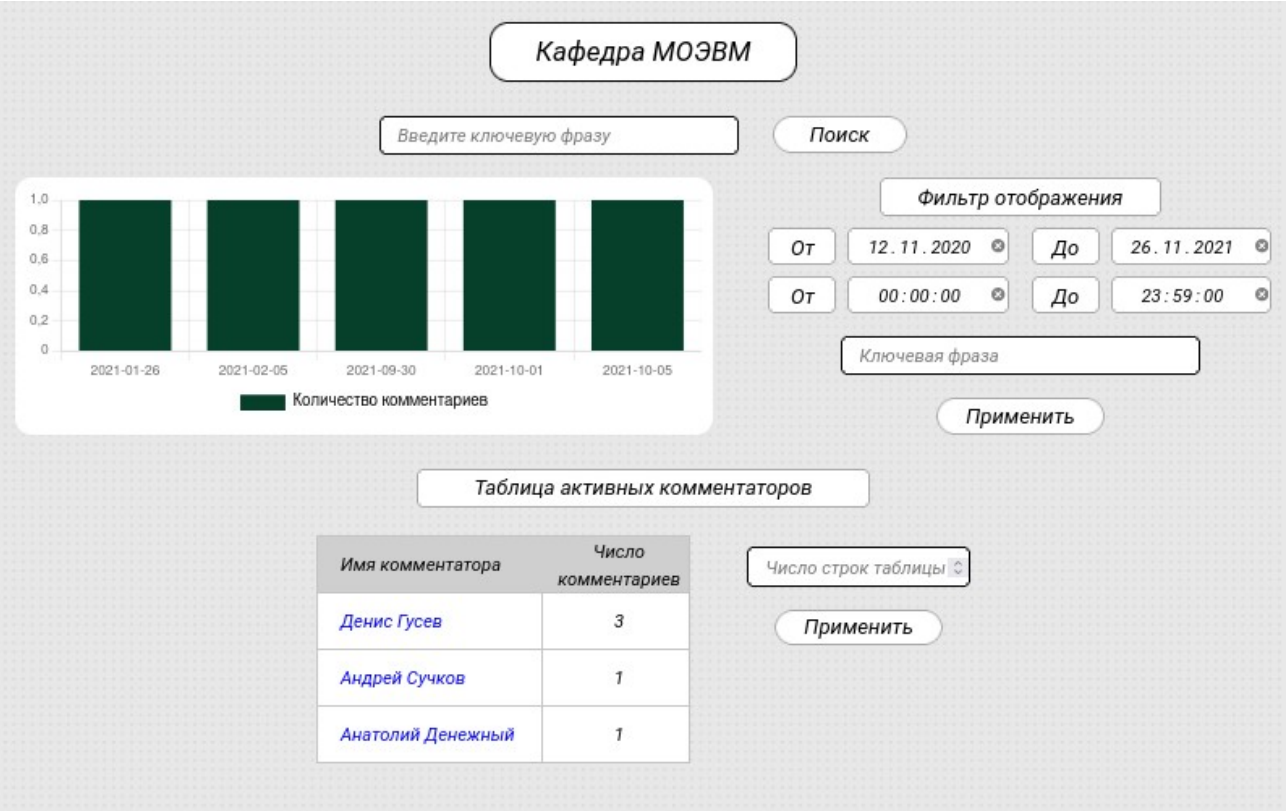

Рисунок 8. Страница группы

Имя комментатора	Время	Дата	Комментарии	Количество лайков
Денис Гусев	13:39:22	30.09.2021	Да, на самом деле с большой благодарностью слушаю лекции Кирилла Владимировича Кринкина!	0
Андрей Сучков	14:36:15	05.02.2021	Подтверждаю: генератор задач работает исправно😊👍🏻	0
Денис Гусев	14:13:51	05.10.2021	[club128046938|Кафедра МОЭВМ], и Вам добрый день! Огромное Вам спасибо! 😊 Очень Вам благодарен, и признателен за то, что Вы уделили время моей просьбе!😊 😊😊 Конечно, это очень полезно для меня. Меня очень заинтересовало системное программирование на языке С в ОС Linux. Хочу основательно разобраться в этом вопросе, и набраться опыта программирования. В этой связи мне стал интересен и ассемблер.😊 Так что очень Вам благодарен! Спасибо.😊	1

Рисунок 9. Результат поиска по ключевой фразе

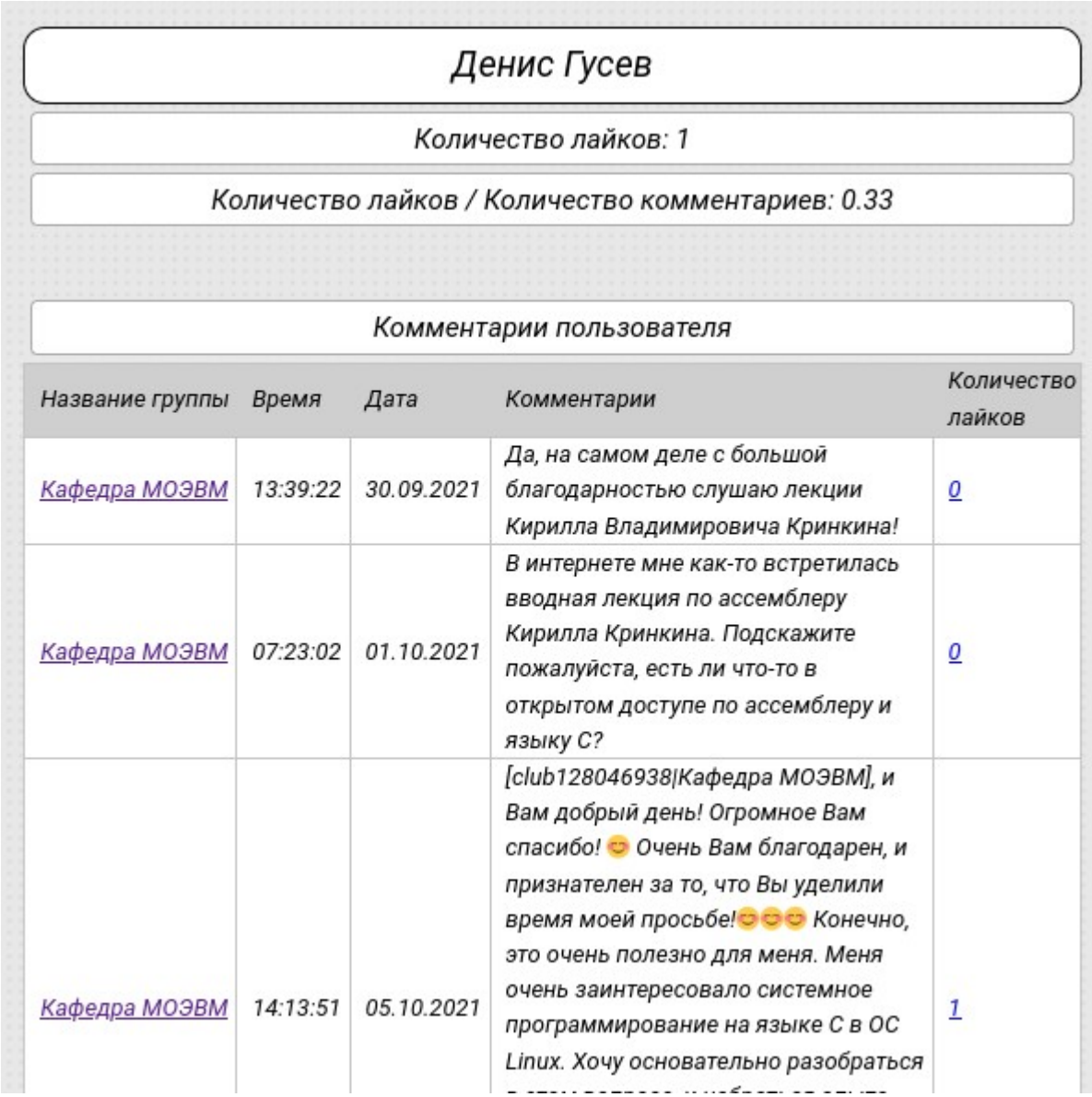

Рисунок 10. Страница пользователя

Рисунок 11. Страница отладки

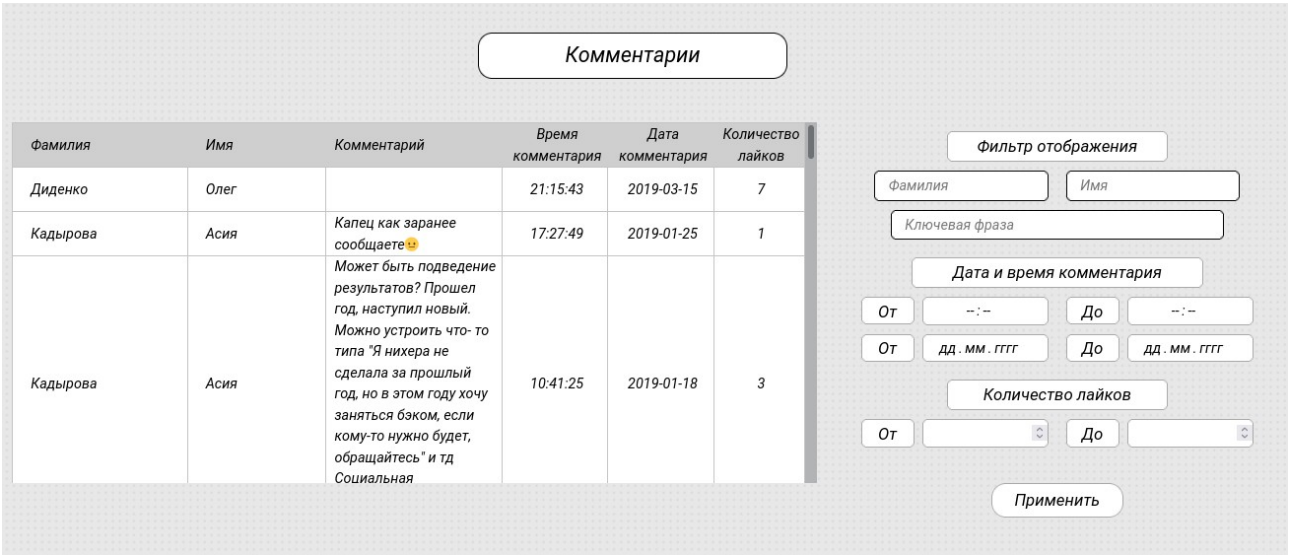

Рисунок 12. Страница отладки комментариев

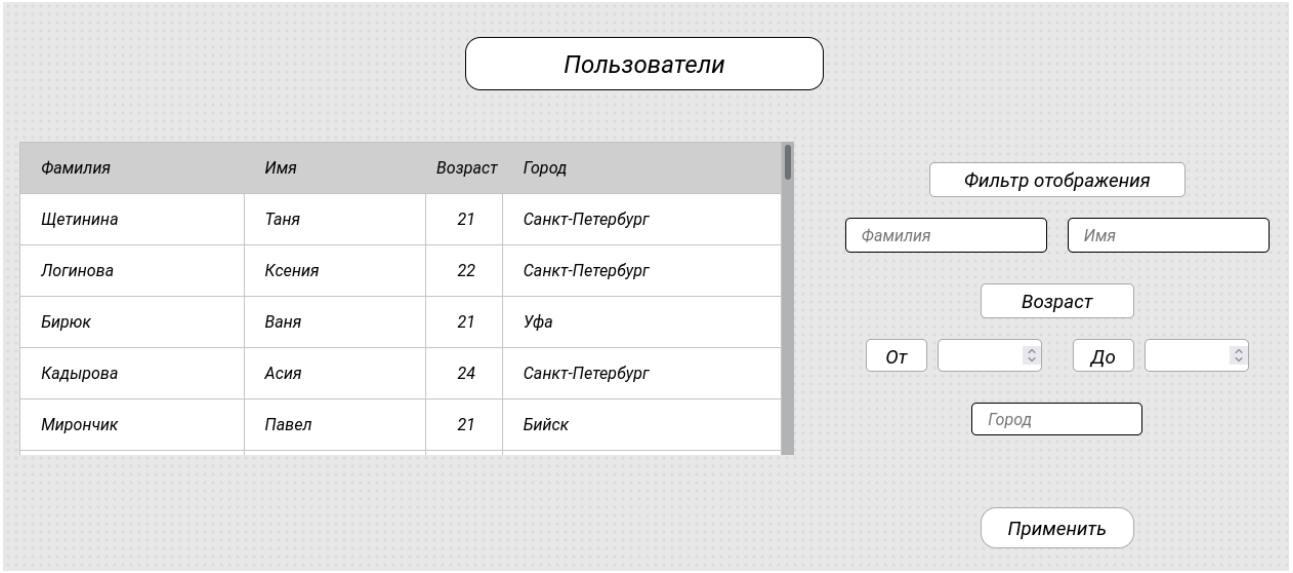

Рисунок 13. Страница отладки пользователей

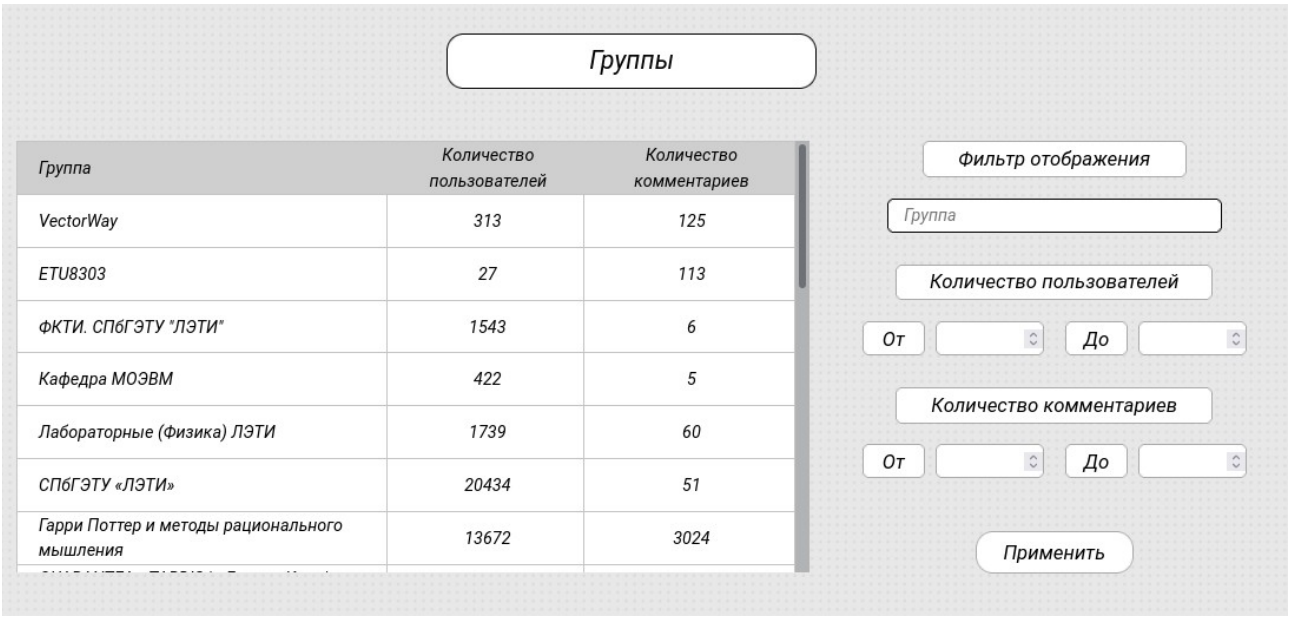

Рисунок 14. Страница отладки групп

7. ИСПОЛЬЗОВАННЫЕ МАТЕРИАЛЫ

1. Документация MongoDB // MongoDB URL: https://docs.mongodb.com/ (Дата обращения: 27.11.2021)

2. Документация React // React URL: https://reactjs.org/docs/getting-started.html (Дата обращения: 27.11.2021)

3. Документация ExpressJS // ExpressJS URL: https://expressjs.com/ru/4x/api.html (Дата обращения: 27.11.2021)